Contents

[Azure sphere 1](#_Toc35430202)

[A recap – deep unsupervised feature extraction with a classifier on top 3](#_Toc35430203)

[Deep Generative networks 3](#_Toc35430204)

[Restricted Boltzmann machine 3](#_Toc35430205)

[Deep Belief Networks 3](#_Toc35430206)

[Autoencoders 4](#_Toc35430207)

[**Anomaly Detection with Autoencoders Made Easy** 4](#_Toc35430208)

[Introduction to autoencoders. 7](#_Toc35430209)

[Jeremy Jordan 8](#_Toc35430210)

[Summary 10](#_Toc35430211)

[Variational autoencoders 11](#_Toc35430212)

[Generative Adversarial Networks 12](#_Toc35430213)

[**A Beginner's Guide to Generative Adversarial Networks (GANs)** 12](#_Toc35430214)

[Generative Adversarial Network Definition 12](#_Toc35430215)

[Generative vs. Discriminative Algorithms 13](#_Toc35430216)

[How GANs Work 14](#_Toc35430217)

[GANs, Autoencoders and VAEs 16](#_Toc35430218)

[Group Exercise 17](#_Toc35430219)

# Azure sphere

* **A microcontroller (MCU for microcontroller unit)** is a small computer on a single metal-oxide-semiconductor (MOS) integrated circuit chip.
* less sophisticated than, a system on a chip **(SoC);** a SoC may include a microcontroller as one of its components.
* A microcontroller contains one or more **CPUs (processor cores) along with memory and programmable input/output peripherals.**
* **Program memory in the form of ferroelectric RAM**, NOR flash or OTP ROM is also often included on chip, as well as a small amount of RAM.
* **Microcontrollers are designed for embedded applications**, in contrast to the microprocessors used in personal computers or other general purpose applications consisting of various discrete chips.
* Microcontrollers are used in **automatically controlled products and devices**, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems.
* **By reducing the size and cost compared to a design** that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes.

Examples:

• ARM core processors (many vendors)

o ARM Cortex-M cores are specifically targeted toward microcontroller applications

• Microchip Technology Atmel AVR (8-bit), AVR32 (32-bit), and AT91SAM (32-bit)

• Cypress Semiconductor's M8C core used in their PSoC (Programmable System-on-Chip)

• Freescale ColdFire (32-bit) and S08 (8-bit)

• Freescale 68HC11 (8-bit), and others based on the Motorola 6800 family

• Intel 8051, also manufactured by NXP Semiconductors, Infineon and many others

• Infineon: 8-bit XC800, 16-bit XE166, 32-bit XMC4000 (ARM based Cortex M4F), 32-bit TriCore and, 32-bit Aurix Tricore Bit microcontrollers[34]

• Maxim Integrated MAX32600, MAX32620, MAX32625, MAX32630, MAX32650, MAX32640

• MIPS

• Microchip Technology PIC, (8-bit PIC16, PIC18, 16-bit dsPIC33 / PIC24), (32-bit PIC32)

• NXP Semiconductors LPC1000, LPC2000, LPC3000, LPC4000 (32-bit), LPC900, LPC700 (8-bit)

• Parallax Propeller

• PowerPC ISE

• Rabbit 2000 (8-bit)

• Renesas Electronics: RL78 16-bit MCU; RX 32-bit MCU; SuperH; V850 32-bit MCU; H8; R8C 16-bit MCU

• Silicon Laboratories Pipelined 8-bit 8051 microcontrollers and mixed-signal ARM-based 32-bit microcontrollers

• STMicroelectronics STM8 (8-bit), ST10 (16-bit), STM32 (32-bit), SPC5 (automotive 32-bit)

• Texas Instruments TI MSP430 (16-bit), MSP432 (32-bit), C2000 (32-bit)

• Toshiba TLCS-870 (8-bit/16-bit)

* **Pete warden tinyml and others**
* **What is Azure sphere** - <https://www.youtube.com/watch?v=1NT4m2kHYE4&t=541s>
* **Demo**

# A recap – deep unsupervised feature extraction with a classifier on top

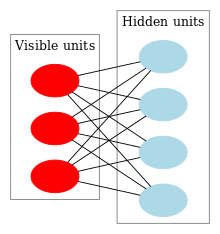
# Deep Generative networks

Deep generative networks: (vs discriminative networks, vs deep discriminative networks)

DBN, Deep Boltzmann Machine (DBM), Generative Adversarial Network (GAN), and Variational Autoencoder (VAE) are discussed

## Restricted Boltzmann machine

* Restricted Boltzmann machine (RBM) is a general mechanism (building block) which has no connections between hidden units and can be trained efficiently.
* Multiple hidden layers can be learned by treating the hidden layer output of one RBM as the data for training a higher-level RBM.



<https://pathmind.com/wiki/restricted-boltzmann-machine>

## Deep Belief Networks

<https://en.wikipedia.org/wiki/Deep_belief_network>

* In machine learning, a **deep belief network (DBN)** is a generative graphical model, or alternatively a class of deep neural network, composed of multiple layers of latent variables ("hidden units"), **with connections between the layers but not between units within each layer.**
* When trained on a set of examples **without supervision**, a DBN can learn to probabilistically reconstruct its inputs. The layers then act as feature detectors.
* After this learning step, a DBN can be further trained **with supervision to perform classification.**
* DBNs can be viewed as a **composition of simple, unsupervised networks such as restricted Boltzmann machines (RBMs)** where each sub-network's hidden layer serves as the visible layer for the next.
* **An RBM** is an undirected, generative energy-based model with a "visible" input layer and a hidden layer and connections between but not within layers.
* This composition **leads to a fast, layer-by-layer unsupervised training procedure**, where contrastive divergence is applied to each sub-network in turn, starting from the "lowest" pair of layers (the lowest visible layer is a training set).

## Autoencoders

<https://towardsdatascience.com/anomaly-detection-with-autoencoder-b4cdce4866a6>

# **Anomaly Detection with Autoencoders Made Easy**

**Autoencoders Come from Artificial Neural Network**

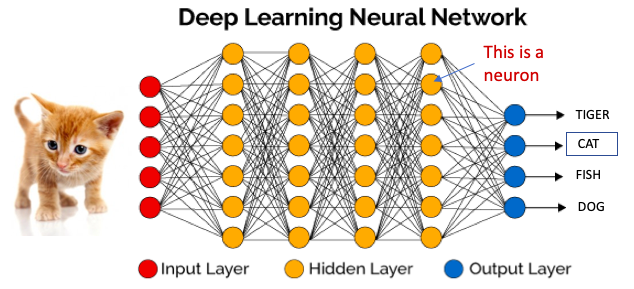


Figure (A): Artificial Neural Network

An autoencoder is a special type of neural network that copies the input values to the output values as shown in Figure (B). It does not require the target variable like the conventional Y, thus is categorized as unsupervised learning. You may ask why we train the model if the output values are set to equal to the input values. Indeed, we are not so much interested in the output layer. We are interested in the hidden core layer. If the number of neurons in the hidden layers is less than that of the input layers, the hidden layers will extract the essential information of the input values. This condition forces the hidden layers to learn the most patterns of the data and ignore the “noises”. So in an autoencoder model, the hidden layers must have less dimensions than those of the input or output layers. If the number of neurons in the hidden layers is more than those of the input layers, the neural network will be given too much capacity to learn the data. In an extreme case, it could just simply copy the input to the output values, including noises, without extracting any essential information.

Figure (B) also shows the encoding and decoding process. The encoding process compresses the input values to get to the core layer. The decoding process reconstructs the information to produce the outcome. The decoding process mirrors the encoding process in the number of the hidden layers and neurons. Most practitioners just adopt this symmetry.

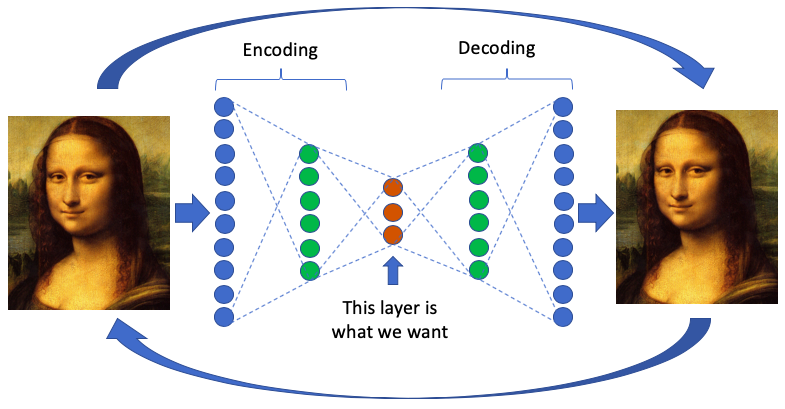
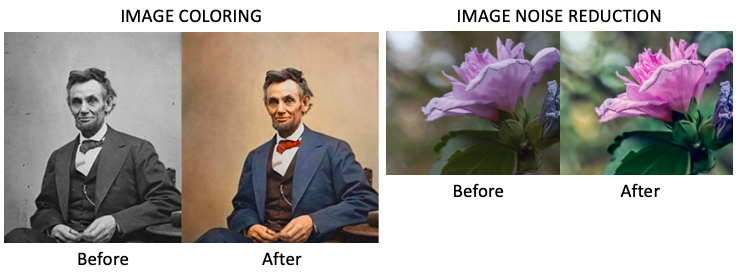


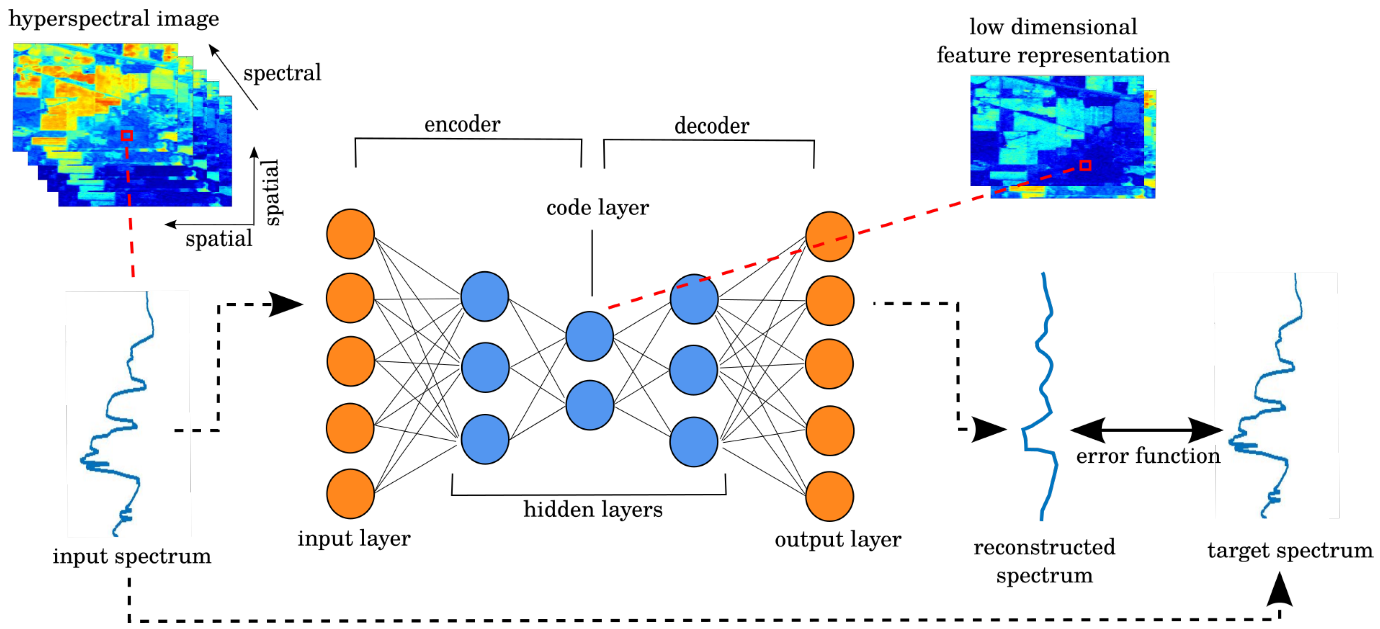
Figure (B): Autoencoders

**What Are the Applications of Autoencoders?**

The early application of autoencoders is dimensionality reduction. A milestone paper by [Geoffrey Hinton (2006)](https://en.wikipedia.org/wiki/Geoffrey_Hinton) showed a trained autoencoder yielding a smaller error compared to the first 30 principal components of a PCA and a better separation of the clusters. Autoencoders also have wide applications in computer vision and image editing. In image coloring, autoencoders are used to convert a black-and-white image to a colored image. In image noise reduction, autoencoders are used to remove nosies. See my post “[Convolutional Autoencoders for Image Noise Reduction](https://medium.com/@Dataman.ai/convolutional-autoencoders-for-image-noise-reduction-32fce9fc1763)”.



<https://www.mdpi.com/2072-4292/11/7/864/htm>



**Why Do We Apply Dimensionality Reduction to Find Outliers?**

Don’t we lose some information, including the outliers, if we reduce the dimensionality? The answer is once the main patterns are identified, the outliers are revealed. Many distance-based techniques (e.g. KNNs) suffer the curse of dimensionality when they compute distances of every data point in the full feature space. High dimensionality have to be reduced. Interestingly, during the process of dimensionality reduction outliers are identified. We can say outlier detection is a by-product of dimension reduction.

**Why Autoencoders?**

There are already many useful tools such as Principal Component Analysis (PCA) to detect outliers, why do we need the autoencoders? Recall that the PCA uses linear algebra to transform (see this article “[Dimension Reduction Techniques with Python](https://towardsdatascience.com/dimension-reduction-techniques-with-python-f36ca7009e5c)”). In contrast, the autoencoder techniques can perform non-linear transformations with their non-linear activation function and multiple layers. It is more efficient to train several layers with an autoencoder, rather than training one huge transformation with PCA. The autoencoder techniques thus show their merits when the data problems are complex and non-linear in nature.

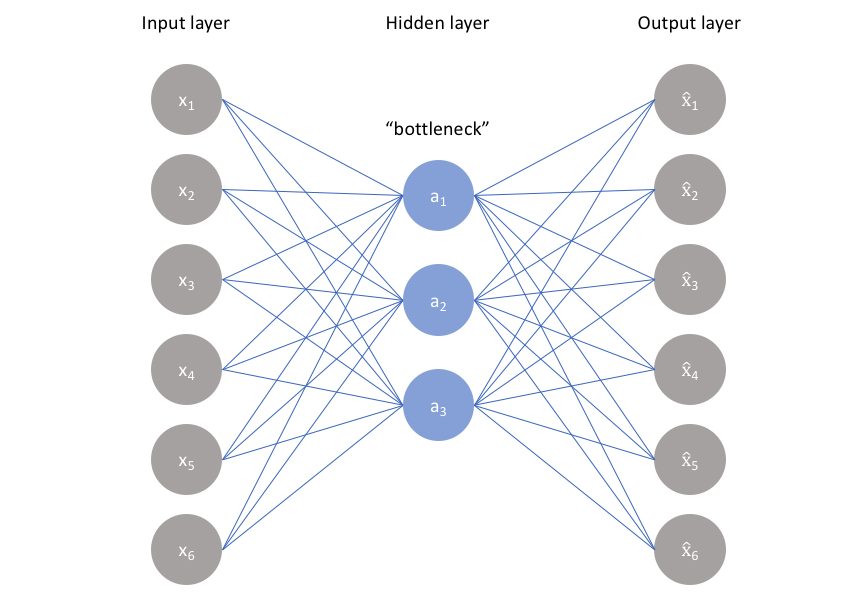
I hope the above briefing motivates you to apply the autoencoder algorithm for outlier detection. Let’s build the model now. If you want to know more about the Artificial Neural Networks (ANN), please watch the video clip below. If you are comfortable with ANN, you can move on to the Python code.

<https://www.jeremyjordan.me/autoencoders/>

# Introduction to autoencoders.

## Jeremy Jordan

Autoencoders are an unsupervised learning technique in which we leverage neural networks for the task of **representation learning**. Specifically, we'll design a neural network architecture such that we impose a bottleneck in the network which forces a ***compressed*** knowledge representation of the original input. If the input features were each independent of one another, this compression and subsequent reconstruction would be a very difficult task. However, if some sort of structure exists in the data (ie. correlations between input features), this structure can be learned and consequently leveraged when forcing the input through the network's bottleneck.

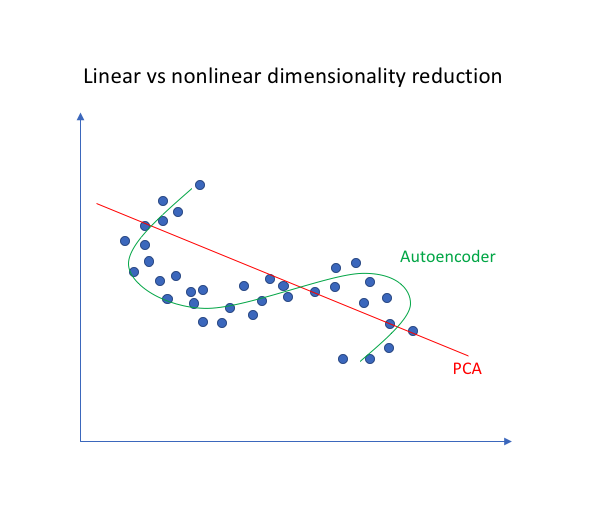


The ideal autoencoder model balances the following:

* Sensitive to the inputs enough to accurately build a reconstruction.
* Insensitive enough to the inputs that the model doesn't simply memorize or overfit the training data.

This trade-off forces the model to maintain only the variations in the data required to reconstruct the input without holding on to redundancies within the input.

Because neural networks are capable of learning nonlinear relationships, this can be thought of as a more powerful (nonlinear) generalization of [PCA](https://www.jeremyjordan.me/principal-components-analysis/). Whereas PCA attempts to discover a lower dimensional hyperplane which describes the original data, autoencoders are capable of [learning nonlinear manifolds](http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/) (a manifold is defined in simple terms as a continuous, non-intersecting surface). The difference between these two approaches is visualized below.

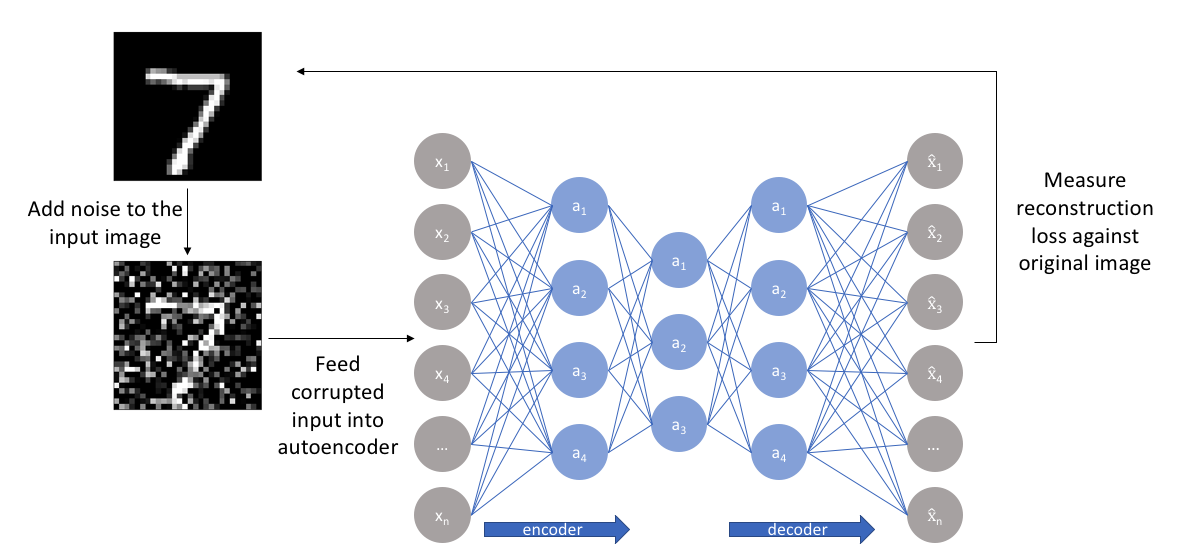


#### Sparse autoencoders

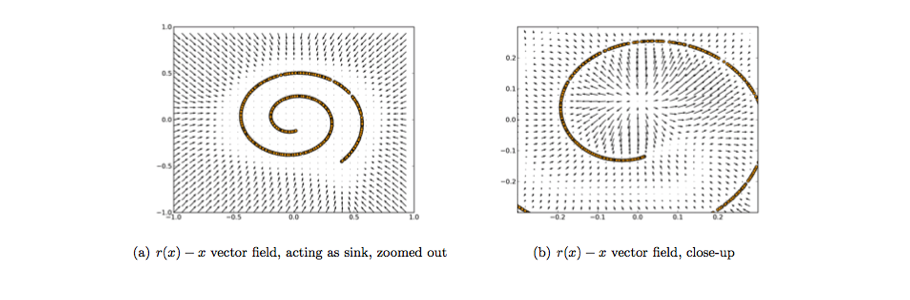
Sparse autoencoders offer us an alternative method for introducing an information bottleneck without requiring a reduction in the number of nodes at our hidden layers. Rather, we'll construct our loss function such that we penalize activations within a layer. For any given observation, we'll encourage our network to learn an encoding and decoding which only relies on activating a small number of neurons.

#### Denoising autoencoders

So far I've discussed the concept of training a neural network where the input and outputs are identical and our model is tasked with reproducing the input as closely as possible while passing through some sort of information bottleneck. Recall that I mentioned we'd like our autoencoder to be sensitive enough to recreate the original observation but insensitive enough to the training data such that the model learns a generalizable encoding and decoding. Another approach towards developing a generalizable model is to slightly corrupt the input data but still maintain the uncorrupted data as our target output.



With this approach, **our model isn't able to simply develop a mapping which memorizes the training data because our input and target output are no longer the same**. Rather, the model learns a vector field for mapping the input data towards a lower-dimensional manifold (recall from my earlier graphic that a manifold describes the high density region where the input data concentrates); if this manifold accurately describes the natural data, we've effectively "canceled out" the added noise.



## Summary

An autoencoder is a neural network architecture capable of discovering structure within data in order to develop a compressed representation of the input. Many different variants of the general autoencoder architecture exist with the goal of ensuring that the compressed representation represents meaningful attributes of the original data input; typically the biggest challenge when working with autoencoders is getting your model to actually learn a meaningful and generalizable latent space representation.

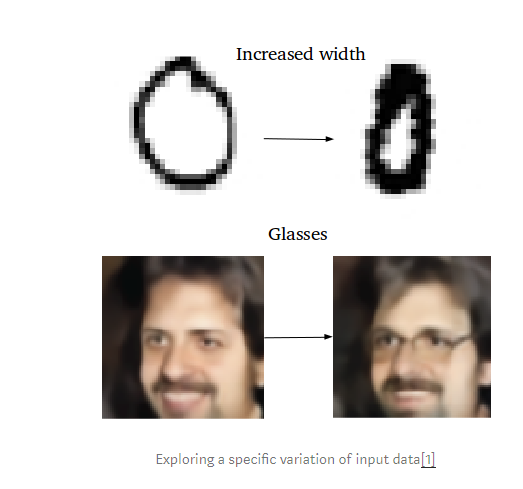
Because autoencoders learn how to compress the data based on attributes (ie. correlations between the input feature vector) discovered from data during training, these models are typically only capable of reconstructing data similar to the class of observations of which the model observed during training.

Applications of autoencoders include:

* Anomaly detection
* Data denoising (ex. images, audio)
* Image inpainting
* Information retrieval

## Variational autoencoders

<https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf>



## Generative Adversarial Networks

<https://pathmind.com/wiki/generative-adversarial-network-gan>

# **A Beginner's Guide to Generative Adversarial Networks (GANs)**

You might not think that programmers are artists, but programming is an extremely creative profession. It’s logic-based creativity. - John Romero

## Generative Adversarial Network Definition

Generative adversarial networks (GANs) are algorithmic architectures that use two neural networks, pitting one against the other (thus the “adversarial”) in order to generate new, synthetic instances of data that can pass for real data. They are used widely in image generation, video generation and voice generation.

[GANs were introduced in a paper](https://arxiv.org/abs/1406.2661) by Ian Goodfellow and other researchers at the University of Montreal, including Yoshua Bengio, in 2014. Referring to GANs, Facebook’s AI research director Yann LeCun [called adversarial training](https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning) “the most interesting idea in the last 10 years in ML.”

GANs’ potential for both good and evil is huge, because they can learn to mimic any distribution of data. That is, GANs can be taught to create worlds eerily similar to our own in any domain: images, music, speech, prose. They are robot artists in a sense, and their [output is impressive](https://www.nytimes.com/2017/08/14/arts/design/google-how-ai-creates-new-music-and-new-artists-project-magenta.html) – poignant even. But they can also be used to generate fake media content, and are the technology underpinning [Deepfakes](https://en.wikipedia.org/wiki/Deepfake).

In a surreal turn, [Christie’s sold a portrait](https://www.theverge.com/2018/10/23/18013190/ai-art-portrait-auction-christies-belamy-obvious-robbie-barrat-gans) for $432,000 that had been generated by a GAN, based on [open-source code written by Robbie Barrat of Stanford](https://github.com/robbiebarrat/art-DCGAN). Like most true artists, he didn’t see any of the money, which instead went to the French company, Obvious.[0](https://pathmind.com/wiki/generative-adversarial-network-gan#zero)

In 2019, DeepMind showed that [variational autoencoders (VAEs) could outperform GANs on face generation](https://syncedreview.com/2019/06/06/going-beyond-gan-new-deepmind-vae-model-generates-high-fidelity-human-faces/).

## Generative vs. Discriminative Algorithms

To understand GANs, you should know how generative algorithms work, and for that, contrasting them with discriminative algorithms is instructive. Discriminative algorithms try to classify input data; that is, given the features of an instance of data, they predict a label or category to which that data belongs.

For example, given all the words in an email (the data instance), a discriminative algorithm could predict whether the message is spam or not\_spam. spam is one of the labels, and the bag of words gathered from the email are the features that constitute the input data. When this problem is expressed mathematically, the label is called y and the features are called x. The formulation p(y|x) is used to mean “the probability of y given x”, which in this case would translate to “the probability that an email is spam given the words it contains.”

So discriminative algorithms map features to labels. They are concerned solely with that correlation. One way to think about generative algorithms is that they do the opposite. Instead of predicting a label given certain features, they attempt to predict features given a certain label.

The question a generative algorithm tries to answer is: Assuming this email is spam, how likely are these features? While discriminative models care about the relation between y and x, generative models care about “how you get x.” They allow you to capture p(x|y), the probability of x given y, or the probability of features given a label or category. (That said, generative algorithms can also be used as classifiers. It just so happens that they can do more than categorize input data.)

Another way to think about it is to distinguish discriminative from generative like this:

* Discriminative models learn the boundary between classes
* Generative models model the distribution of individual classes

[Learn to build AI apps now »](https://skymind.ai/learn)

## How GANs Work

One neural network, called the generator, generates new data instances, while the other, the discriminator, evaluates them for authenticity; i.e. the discriminator decides whether each instance of data that it reviews belongs to the actual training dataset or not.

Let’s say we’re trying to do something more banal than mimic the Mona Lisa. We’re going to generate hand-written numerals like those found in the MNIST dataset, which is taken from the real world. The goal of the discriminator, when shown an instance from the true MNIST dataset, is to recognize those that are authentic.

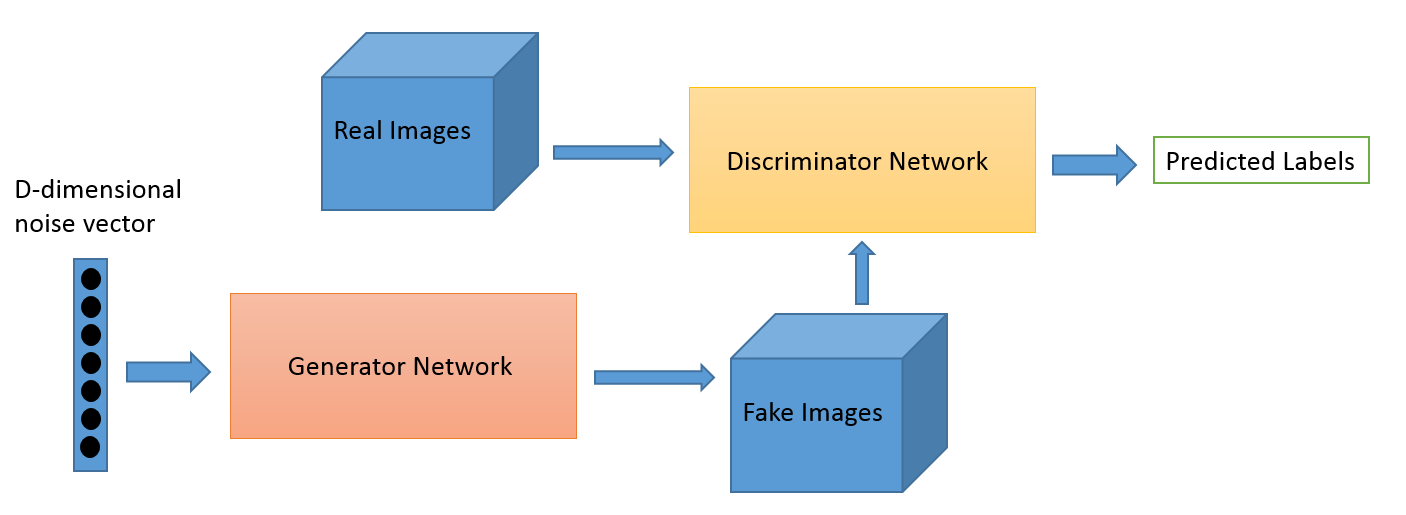
Meanwhile, the generator is creating new, synthetic images that it passes to the discriminator. It does so in the hopes that they, too, will be deemed authentic, even though they are fake. The goal of the generator is to generate passable hand-written digits: to lie without being caught. The goal of the discriminator is to identify images coming from the generator as fake.

Here are the steps a GAN takes:

* The generator takes in random numbers and returns an image.
* This generated image is fed into the discriminator alongside a stream of images taken from the actual, ground-truth dataset.
* The discriminator takes in both real and fake images and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.

So you have a double feedback loop:

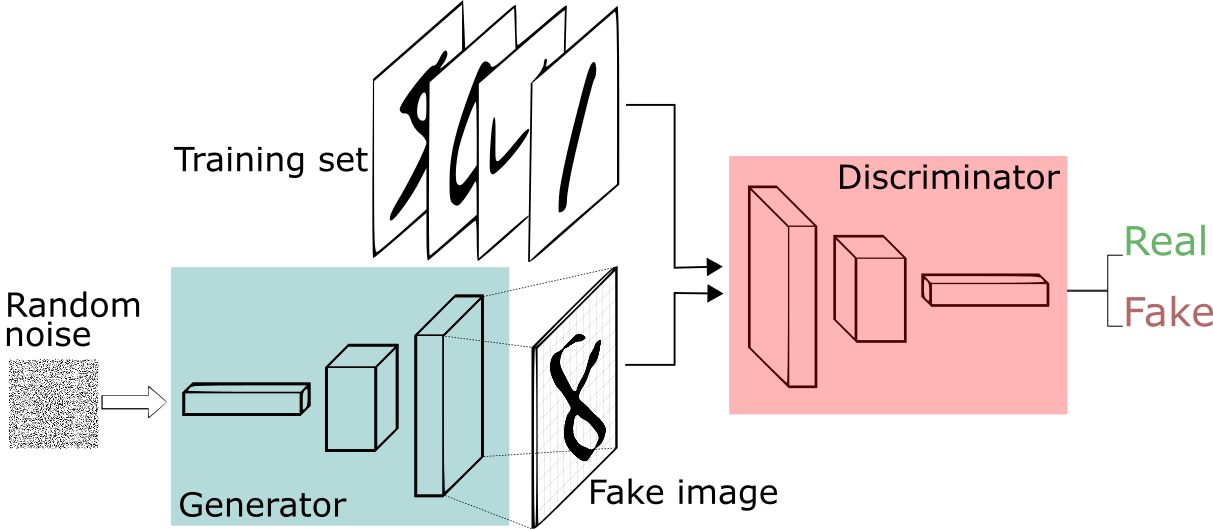
* The discriminator is in a feedback loop with the ground truth of the images, which we know.
* The generator is in a feedback loop with the discriminator.

 Credit: O’Reilly

You can think of a GAN as the opposition of a counterfeiter and a cop in a game of cat and mouse, where the counterfeiter is learning to pass false notes, and the cop is learning to detect them. Both are dynamic; i.e. the cop is in training, too (to extend the analogy, maybe the central bank is flagging bills that slipped through), and each side comes to learn the other’s methods in a constant escalation.

For MNIST, the discriminator network is a standard convolutional network that can categorize the images fed to it, a binomial classifier labeling images as real or fake. The generator is an inverse convolutional network, in a sense: While a standard convolutional classifier takes an image and downsamples it to produce a probability, the generator takes a vector of random noise and upsamples it to an image. The first throws away data through downsampling techniques like maxpooling, and the second generates new data.

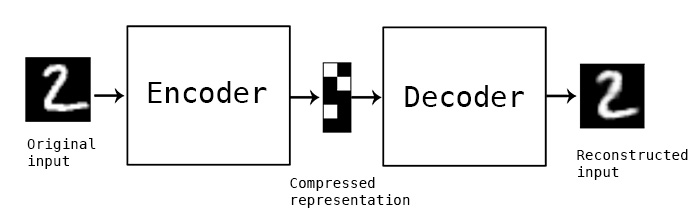
Both nets are trying to optimize a different and opposing objective function, or loss function, in a zero-zum game. This is essentially an [actor-critic model](https://arxiv.org/abs/1610.01945). As the discriminator changes its behavior, so does the generator, and vice versa. Their losses push against each other.

 Image credit: *[Thalles Silva](https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394)*

If you want to learn more about generating images, Brandon Amos wrote a great post about [interpreting images as samples from a probability distribution](https://bamos.github.io/2016/08/09/deep-completion/#step-1-interpreting-images-as-samples-from-a-probability-distribution).

## GANs, Autoencoders and VAEs

It may be useful to compare generative adversarial networks to other neural networks, such as autoencoders and variational autoencoders.

Autoencoders encode input data as vectors. They create a hidden, or compressed, representation of the raw data. They are useful in dimensionality reduction; that is, the vector serving as a hidden representation compresses the raw data into a smaller number of salient dimensions. Autoencoders can be paired with a so-called decoder, which allows you to reconstruct input data based on its hidden representation,  [Credit: Keras blog](https://blog.keras.io/building-autoencoders-in-keras.html)

Variational autoencoders are generative algorithm that add an additional constraint to encoding the input data, namely that the hidden representations are normalized. Variational autoencoders are capable of both compressing data like an autoencoder and synthesizing data like a GAN. However, while GANs generate data in fine, granular detail, images generated by VAEs tend to be more blurred.

You can bucket generative algorithms into one of three types:

* Given a label, they predict the associated features (Naive Bayes)
* Given a hidden representation, they predict the associated features (VAE, GAN)
* Given some of the features, they predict the rest (inpainting, imputation)

Aut

Autoencoders

<https://stats.stackexchange.com/questions/95428/deep-belief-networks-or-deep-boltzmann-machines>

<https://en.wikipedia.org/wiki/Autoencoder>

<https://towardsdatascience.com/anomaly-detection-with-autoencoder-b4cdce4866a6>

<https://towardsdatascience.com/autoencoder-neural-network-for-anomaly-detection-with-unlabeled-dataset-af9051a048>

variational auto encoders

GANs

A [restricted Boltzmann machine](https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine) (RBM) with fully connected visible and hidden units. Note there are no hidden-hidden or visible-visible connections

# Group Exercise

**6. Cyber Applications of Deep Learning Methods**

6.1. Malware

6.2. Domain Generation Algorithms and Botnet Detection

6.3. Drive-By Download Attacks

6.4. Network Intrusion Detection

6.5. File Type Identification

6.6. Network Traffic Identification

6.7. SPAM Identification

6.8. Insider Threat Detection

6.9. Border Gateway Protocol Anomaly Detection

6.10. Verification If Keystrokes Were Typed by a Human

6.11. User Authentication

6.12. False Data Injection Attack Detection

IoT